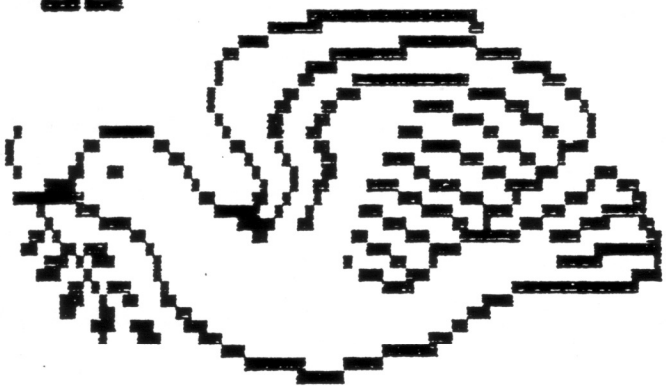
VZ-LINK MAGAZINE

NEW ZEALAND

VZ 300/200

COMPUTER USERS CLUB

ISSUE 33



JULY 89

COMMENT

Welcome to another Issue of YOUR newsletter.I hope I have included something to interest everyone.

This month We start a new series which covers or is it Uncovers the mysteries of VZ Dos.

I would once again like to thank Mark Harwood for giving permission to reprint this very fine series.

I would also like to Thank Ken Clarke for his input for the newsletter.

Ken is also working on some other interesting aid's (if I may use that word) to VZ users.

The next workshop will be held on Saturday 12 AUGUST.at 1030 am.
Details in the next issue.

There is still something missing at these workshops,
If you have any ideas or suggestions please let me know.
I do hope you can all attend these workshops.

Until next issue,have fun and remember those newsletter items.

*** VZ-200/300 MAILING LIST ***

"DISKLIST" - Conversion to Disk - Update.

by Ken Clarke.

I am very impressed with the results I got from Joe Leon's two articles appearing in 'NEWSLINK' N°'s 24 & 25 (Sept/Oct. 1988). I offer the results of my efforts to further enhance his work.

```
1200 COLOR7:PRINT@450,"| WHICH FUNCTION <0-9> ? |      ";:SOUND30,1

2130 IFA$(1)<>""THENGOSUB300ELSEDT=DT-1:GOSUB40000:GOTO1010

27 PRINT@418,"| PRESS <<SPACE>>    X:INFORM |";:K$=INKEY$:GOTO29

29 I$=INKEY$:IFI$=" "THENRETURNELSEIFI$="X"THEN61000ELSE29

33 A1=PEEK(30897)+PEEK(30898)*256-45000

 35 CLS:CLEAR  A1 :Q$=CHR$(34)

 1230 IFI$<"0"ORI$>"9"THEN1210ELSESOUND30,1
```

There are two tape versions, and this can be noted by looking at line 20 (as Joe points out). His conversion was on the version which apparently does not adjust the cleared area according to the amount of memory available, by not having line 33 and line 35 including -CLEAR 12000- instead of that above. My work was on the opposite number. The first problem was an embarrasing message 'OUT OF MEMORY' which brought about the first modification of changing line 33 from .... 43000 to ....45000. It is an advantage to be able to make the best use of all available ram, and this can be done with lines 33 & 35. (Remember to adjust the GOTO number - see original article)

Mod.2. Does the version converted by Joe have a number of screens of instruction (line numbers 61000-62060)? To gain access to these, I modified lines 27 & 29 as shown. On the initial RUN there is now the option to view this information by pressing X

Mod.3. Joe's update to give a CLEAR FILE option can be given a little cosmetic job by altering line 12ØØ to read 'Ø-9' instead of '1-9'. This corrects the menu screen.

Mod.4. A hic-up occured when in the MAIN MENU mode. Pressing a letter key gives a lock-up. The cure is to alter line 123Ø as I have shown.

Mod.5. Another cosmetic touch up to line 213Ø will reprint the title 'MAIN MENU' - when quitting from the entering data mode. As with the previous mod' it is only necessary to change the GOTO number.

==========

Need HELP to get your conversion working? See you at a Workshop Meeting or leave a message with Peter.

==========

INTERRUPTS

PART TWO

BY BOB KITCH.

THE INTERRUPT VECTOR.

Considerable discussion about the Interrupt Vector located at 787DH in RAM has been made. It is possible for the user to write system interrupt routines. The single interrupt generator is the VDG (as mentioned previously), which initiates an interrupt every 20 msec. for the Z80A CPU. This is introduced via the *INT pin on the Z80. This causes the Maskable Interrupt Mode in the CPU. Note that since the interrupt signal is the negative or down going edge of the vertical sync. pulse of the VDG, the Video RAM should only be accessed during the sync. pulse in the interrupt program. The Video RAM should not be accessed at other times. The Z80 is set up to JUMP to 0038H upon receipt of the *INT line going negative. The interrupt service routine, resident in ROM is thereby entered. During the course of this latter routine control is passed out to the Interrupt Vector located from 787DH to 787FH. At this location the vector may be stolen.

It is useful to provide a disassembled outline of this procedure as many users will wish to play around with this facility.

```
0038   JP 2EB8        ;To Interrupt Service Routine

2EB8   PUSH AF        ;Interrupt Service Routine
       PUSH BC
       PUSH DE
       PUSH HL
       CALL 787D      ;Interrupt Vector
       CALL SCREEN    ;Update screen
       CALL CURSOR    ;Flash cursor
       CALL KEYBRD    ;Check keyboard
       CALL BUZZER    ;Beep if keypressed
       POP HL
       POP DE
       POP BC
       POP AF
       EI             ;Enable interrupts
       RETI           ;End of interrupt service
```

A recent program published in ETI used the technique of borrowing the Interrupt Vector. A modified version of the program for a "REAL TIME CLOCK" on the VZ is provided.

DISCUSSION OF REAL TIME CLOCK PROGRAM.

The program provided loads a section of machine code and allocates storage for the time-keeping parameters. Furthermore, the routine is completely relocatable in memory. This latter feature adds some complexity to the program but try to follow what is happening as it actually modifies the machine code routine and is an extremely versatile method.

Subroutines 30, 40 and 50 are quite useful and generalized numeric routines, employed when signed integers (for describing memory locations) are used in a loading routine.

Top of memory is lowered in the familiar manner in lines 210 to 230 and 99 bytes of memory reserved. Following this action, all pointers are reset and variables declared prior to dimensioning. This signicantly speeds up execution as array VA% does not have to be relocated as new variables are found. Integer storage is used throughout to also speed up execution.

The machine code, set out in lines 3020 to 3430, is loaded from lines 310 to 340. The addition of a Checksum variable ensures that incorrect data is detected. Try and follow the "psuedo-assembler" provided in the DATA statements. It is a very simple program.

Lines 410 to 570 use array VA% to modify the addresses in the time keeping routine, thereby making it relocatable. The data contained in lines 4010 to 4300 are the offsets used to modify the routine. Lines 610 and 620 poke the starting address of the routine into the Interrupt Vector.

Line 630 is "REMarked" out but is a valuable Debug Dump of the procedure. Also the dump can provide some insight into the methodology of the program. The code in lines 20020 to 20120 sends the dump to the printer.

The current time is set in lines 710 to 750. Range checking is carried out to ensure predictable program execution.

Finally, the Interrupt Vector is "stolen" in line 195 and, 50 times per second, the routine is entered quite transparently to the user.

Lines 910 to 980 are a looping section of the program that continuously displays the time at the bottom of the screen. There is considerable scope to vary this display section. Line 950 is "REMarked" out and will provide a clear screen for the time display. Another option is suggested in line 3420. Whenever the time is updated, the routine exits through this RETurn. If the time is not updated, the exit is via the conditional RETurn in line 3040. A suitable screen updating display driver could be appended after line 3430 thereby rendering the BASIC driver un-necessary. Try and develope a suitable driver that will update the screen every second and be transparent to any other use of the screen.

As this program was developed on a disk system, the inclusion of lines 10000 and 10010 are used to easily update the disk as development proceeds. Simply enter RUN 10000 every so often to save the latest version.

```
1 '***************************
2 '***  INTERRUPT  DRIVEN  ***
3 '***  REAL  TIME  CLOCK  ***
4 '***  FOR  VZ  COMPUTER  ***
5 '***    BY  BOB KITCH    ***
6 '***TIME KEEPING  ROUTINE***
7 '***    IS COMPLETELY    ***
8 '***     RELOCATABLE     ***
9 '***************************
10 '
20 GOTO 100
26 '
27 '***USEFUL NUMERIC LOADING SUB-ROUTINES.
28 '
29 '***CONVERT UNSIGNED TO SIGNED DECIMAL - PASSED IN UD & SD%.
30 IF UD>32767 THEN SD%=INT(UD-65536) ELSE SD%=INT(UD)
35 RETURN
38 '
39 '***CONVERT UNSIGNED DECIMAL TO MSB & LSB - IN UD, MS%, LS%.
40 MS%=INT(UD/256):LS%=INT(UD-256*MS%)
45 RETURN
48 '
49 '***CONVERT SIGNED TO UNSIGNED DECIMAL - PASSED IN SD% & UD.
50 IF SD%<0 THEN UD=SD%+65536 ELSE UD=SD%
55 RETURN
98 '
99 '***PUT UP INTRO MESSAGE.
100 CLS:PRINT@4,"REAL TIME CLOCK ROUTINE"
110 PRINT@197,"LOADING MACHINE CODE"
199 '
200 '***LOWER TOM TO ENABLE ROUTINE TO BE STORED.
210 TM=(256*PEEK(30898)+PEEK(30897))-100 :'***RESERVE 99 BYTES.
220 UD=TM:GOSUB 40:POKE 30898,MS%:POKE 30897,LS%
230 CLEAR 100                            :'***RESET POINTERS.
240 S%=0:M%=0:H%=0:SA%=0:MA%=0:HA%=0
250 UD=0:TM=0:AD=0:SD%=0:MS%=0:LS%=0     :'***INITIALIZE STORAGE
260 I%=0:J%=0:DV%=0:CS%=0                :'***BEFORE DIMEN'ING.
270 DIM VA%(2,23)                        :'***INITIALIZE ARRAY.
280 TM=256*PEEK(30898)+PEEK(30897)       :'***NEW TOM.
299 '
300 '***READ IN TIME KEEPING ROUTINE.
310 FOR AD=TM+1 TO TM+77
320    UD=AD:GOSUB 30:READ DV%:POKE SD%,DV%
330    CS%=CS%+DV%
340 NEXT AD
350 IF CS%<>8110 THEN PRINT@293,"CHECKSUM NOT CORRECT";CS%:STOP
399 '
400 '***INITIALIZE POINTERS IN VA%().
410 FOR I%=0 TO 23
420    READ VA%(0,I%):UD=TM+VA%(0,I%):GOSUB 30:GOSUB 40
430    VA%(0,I%)=SD%:VA%(1,I%)=LS%:VA%(2,I%)=MS%
440 NEXT I%
499 '
500 '***MODIFY DUMMY ADDRESSES IN TIME KEEPING ROUTINE.
510 POKE VA%(0,4),VA%(1,0):POKE VA%(0,5),VA%(2,0):'***<KOUNT>
520 FOR I%= 1 TO 3                          :'***<SEC><MIN><HOUR>
530    FOR J%=I%*6 TO I%*6+4 STEP 2         :'***3 OCC'ENCES EACH
540       POKE VA%(0,J%),VA%(1,I%):POKE VA%(0,J%+1),VA%(2,I%)
550    NEXT J%
560 NEXT I%
570 SA%=VA%(0,1):MA%=VA%(0,2):HA%=VA%(0,3):'ADDR FOR SEC.MIN.HR
```

```
599 '
600 '***SET UP JUMP ADDRESS IN INTERRUPT VECTOR.
610 UD=TM+1:GOSUB 40
620 POKE 30846,LS%:POKE 30847,MS%      :'***BYTES 2 & 3.
630 'GO TO 20000                :'***DEBUG JUMP TO PRINTER LISTING.
699 '
700 '***SET TIME.
710 PRINT@293,"***ENTER CURRENT TIME."
720 PRINT@325,"INPUT HOURS";:INPUT DV%
730 IF DV%<1 OR DV%>12 THEN GOTO 720 ELSE POKE HA%,DV%
740 PRINT@389,"INPUT MINS ";:INPUT DV%
750 IF DV%<0 OR DV%>60 THEN GOTO 740 ELSE POKE MA%,DV%
799 '
800 '***STEAL INTERRUPT VECTOR.
810 POKE 30845,195                      :'***JUMP TO TIME ROUTINE.
899 '
900 '***DISPLAY TIME ON BOTTOM LINE OF SCREEN.
910 FOR I%=29120 TO 29151
920    POKE I%,109
930 NEXT I%
940 T$="TIME   ##:##:##"                :'***DISPLAY TEMPLATE.
950 'CLS                                :'***REMOVE REM IF REQ'D.
960 S%=PEEK(SA%):M%=PEEK(MA%):H%=PEEK(HA%)
970 PRINT@484,USING T$;H%,M%,S%;
980 GOTO 960
999 '
2000 '***TIME KEEPING ROUTINE***
2010 '***ALL ADDRESSES SET TO 255 ARE DUMMY PARAMETERS AND ARE
2020 '***RESET FROM BASIC.
2030 '***REGISTERS USED AF, BC, HL.
3000 '
3010 '                  ***COUNT DOWN SECONDS.
3020 DATA 033,255,255:'    LD HL,KOUNT
3030 DATA 053         :'    DEC(HL)
3040 DATA 192         :'    RET NZ
3050 '                  ***SECONDS ROUTINE.
3060 DATA 054,050     :'    LD(HL),50
3070 DATA 058,255,255:'    LD A,(SEC)
3080 DATA 060         :'    INC A
3090 DATA 254,060     :'    CP 60
3100 DATA 040,005     :'    JR Z,L1
3110 DATA 050,255,255:'    LD(SEC),A
3120 DATA 024,056     :'    JR L4
3130 '                  ***MINUTES ROUTINE.
3140 DATA 033,255,255:' L1 LD HL,SEC
3150 DATA 054,000     :'    LD(HL),0
3160 DATA 058,255,255:'    LD A,(MIN)
3170 DATA 060         :'    INC A
3180 DATA 254,060     :'    CP 60
3190 DATA 040,005     :'    JR Z,L2
3200 DATA 050,255,255:'    LD(MIN),A
3210 DATA 024,038     :'    JR L4
3220 '                  ***HOUR ROUTINE.
3230 DATA 033,255,255:' L2 LD HL,MIN
3240 DATA 054,000     :'    LD(HL),0
3250 DATA 001,000,001:'    LD BC,256
3260 DATA 033,042,000:'    LD HL,42
3270 DATA 205,092,052:'    CALL 345CH
3280 DATA 058,255,255:'    LD A,(HOUR)
3290 DATA 060         :'    INC A
3300 DATA 254,013     :'    CP 13
3310 DATA 040,005     :'    JR Z,L3
3320 DATA 050,255,255:'    LD(HOUR),A
3330 DATA 024,011     :'    JR L4
```

```
3340 '                    ***HOUR RESET ROUTINE.
3350 DATA 033,255,255:' L3 LD HL,HOUR
3360 DATA 054,001     :'    LD(HL),1
3370 DATA 024,004     :'    JR L4
3380 DATA 000         :'    DS KOUNT
3390 DATA 000         :'    DS SEC
3400 DATA 000         :'    DS MIN
3410 DATA 000         :'    DS HOUR
3420 '                    ***SCREEN UPDATE ROUTINE HERE.
3430 DATA 201         :' L4 RET
3999 '
4000 '***OFFSETS FOR PARAMETER STORAGE ABOVE ROUTINE.
4010 DATA 73          :'KOUNT
4020 DATA 74          :'SEC
4030 DATA 75          :'MIN
4040 DATA 76          :'HOUR
4200 '***OFFSETS FOR PARAMETERS IN MACHINE CODE ROUTINE.
4210 DATA 2,3         :'<KOUNT>
4220 DATA 9,10        :'<SEC>
4230 DATA 17,18       :'<SEC>
4240 DATA 22,23       :'<SEC>
4250 DATA 27,28       :'<MIN>
4260 DATA 35,36       :'<MIN>
4270 DATA 40,41       :'<MIN>
4280 DATA 54,55       :'<HOUR>
4290 DATA 62,63       :'<HOUR>
4300 DATA 67,68       :'<HOUR>
4310 STOP
4999 '
9999 '***UPDATE DISK FILE.
10000 ERA"RTCLOCK":SAVE"RTCLOCK"
10010 STOP
19999 '
20000 '***DEBUG DUMP TO PRINTER.
20010 '***TO ACTIVATE, TAKE OUT REMARK IN LINE#630.
20020 LPRINT"DEBUG DUMP FROM REAL TIME CLOCK":LPRINT
20030 LPRINT"TOM=",TM:LPRINT
20040 LPRINT"INTERRUPT VECTOR",PEEK(30846),PEEK(30847):LPRINT
20050 LPRINT"PARAMETERS"
20060 FOR I%=0 TO 3
20070    LPRINT I%,VA%(0,I%),VA%(1,I%),VA%(2,I%)
20080 NEXT I%:LPRINT
20090 LPRINT"POINTERS IN VA()"
20100 FOR I%=4 TO 23
20110    LPRINT I%,VA%(0,I%),VA%(1,I%),VA%(2,I%)
20120 NEXT I%:LPRINT
30000 END
```

P.O.BOX 1972 C.P.O AUCKLAND N.Z

(C) AUCKLAND VZ 300/200 COMPUTER USER'S CLUB 1986

ALL MATERIAL IN THIS NEWSLETTER IS SUBJECT TO COPYRIGHT..
CONTRIBUTED PROGRAMS & ARTICLES CAN BE REPRODUCED ON THE UNDERSTANDING THAT THEY ARE FOR THE PRIVATE USE OF SUBSCRIBERS ONLY.
COPYRIGHT IS ALWAY'S RETAINED BY THE AUTHOR.
THIS CLUB WILL NOT BE HELD RESPONSIBLE FOR THE BREECHING OF COPYRIGHT CAUSED BY THE PUBLICATION OF ANY MATERIAL SENT IN UNDER A FALSE NAME.

YOU ARE WELCOME TO PHONE ME IF YOU HAVE PROBLEMS
PHONE 761-451

CUSTOMIZING W.P. BY DAVE BOYCE

```
PRINT TEXT
==========

SCREEN OR PRINTER (S/P)   ?P
RIGHT-JUSTIFY (Y/N)       ?Y
PAGE WIDTH (20-99)        ?31
LEFT MARGIN (0-80)        ?00
PAGE LENGTH (10-99)       ?62
INTER-PAGE LENGTH (0-9)   ?4
AUTO-NEXT-PAGE (Y/N)      ?Y
PAGE NUMBERING (Y/N)      ?N
FIRST PAGE NUMBER (1-99)  ?01
SUPRESS FINAL F-F (Y/N)   ?N
DOUBLE SPACING (Y/N)      ?N
NUMBER OF COPIES (1-9)    ?1
```

```
PRINT TEXT
==========

SCREEN OR PRINTER (S/P)   ?S
RIGHT-JUSTIFY (Y/N)       ?Y
PAGE WIDTH (20-99)        ?75
LEFT MARGIN (0-80)        ?10
PAGE LENGTH (10-99)       ?62
INTER-PAGE LENGTH (0-9)   ?4
AUTO-NEXT-PAGE (Y/N)      ?Y
PAGE NUMBERING (Y/N)      ?N
FIRST PAGE NUMBER (1-99)  ?01
SUPRESS FINAL F-F (Y/N)   ?N
DOUBLE SPACING (Y/N)      ?N
NUMBER OF COPIES (1-9)    ?1
```

```
-27610 80 P
-27609 89 Y
-27608 31
-27607  0
-27606 62 >
-27605  4
-27604 89 Y
-27603 78 N
-27602  1
-27601 78 N
-27600 78 N
-27599  1
```

```
-27610 83 S
-27609 89 Y
-27608 75 K
-27607 10
-27606 62 >
-27605  4
-27604 89 Y
-27603 78 N
-27602  1
-27601 78 N
-27600 78 N
-27599  1
```

```
10 CLS:PRINT:FORL=-27610TO-27599:PRINTTAB(10)L;
20 PRINTUSING"## ";PEEK(L);:IFPEEK(L)<32THENPRINT:GOTO40
30 PRINTCHR$(PEEK(L))
40 NEXT
```

THE E & F WORD PROCESSOR AS SOLD BY DICK SMITH ELECTRONICS WAS AND IS ONE OF THE BETTER TYPE PROGRAMS AVAILABLE FOR THE VZ. MANY VZ USERS, MYSELF INCLUDED FOUND IT ANNOYING HAVING TO SET UP THE PRINT MENU EACH TIME THE W.P. IS LOADED IN THE VZ.

AS EDITOR I USE THE W.P. A FAIR BIT AND THANKS TO DAVE BOYCE WHO SUPPLIED THE INFO ON HOW TO CHANGE THE DEFAULTS THE SOURCE OF ANNOYANCE HAS BEEN REMOVED. TO MODIFY THE W.P. PRINT MENU TO YOUR OWN DEFAULTS IS SIMPLE ENOUGH, EVEN FOR THE BEGINNER. UNFORTUNATELY THE MODS DESCRIBED ARE ONLY FOR DISK DRIVE USERS.

THE REASON IS THAT WHEN W.P. IS CLOADED FROM TAPE IT AUTORUNS AND MODS CAN'T BE CARRIED OUT. AND NOW TO THE MODS. TYPE IN AND SAVE TO DISK THE FOUR LINE MEMORY PEEK PROGRAM. IE :-

SAVE"WP-PEEK"

AT TOP OF THIS PAGE THERE ARE FOUR SCREEN DUMPS, TWO OF THE PRINT MENU AND TWO OF WP-PEEK PROGRAM IN ACTION. THE TWO LEFT ONES ARE BEFORE THE MODS WHILE THE TWO RIGHT ONES ARE AFTER THE MODS.

NEXT RESET THE VZ AND BLOAD W.P. FROM DISK AND THEN RUN"WP-PEEK" PROGRAM. YOU SHOULD SEE ON THE SCREEN A DISPLAY AS PER BOTTOM LEFT SCREEN DUMP PROVIDED YOUR W.P. HAS'NT BEEN MODIFIED PREVIOUSLY.

THE NUMBERS ON THE LEFT REPRESENT MEMORY LOCATIONS OF THE PRINT MENU DEFAULTS, NEXT THE ASCII NUMBERS FOLLOWED BY ALPHABETIC EQUIVALENTS. KNOWING THE MEMORY LOCATIONS WE CAN POKE THE NEW DEFAULTS IN, IT'S AS EASY AS THAT. AND NOW TO THE MODS AND WE'LL TAKE EACH DEFAULT IN TURN.

CUSTOMIZING W.P. CONT. . . . .

SCREEN OR PRINTER (S/P) ?P
POKE -27610,80 FOR PRINTER - POKE -27610,83 FOR SCREEN.

RIGHT-JUSTIFY (Y/N) ?Y
POKE -27609,89 FOR YES - POKE -27609,78 FOR NO

PAGE WIDTH (20-99) ?31
POKE -27608,N - N=20 TO 99 - YOUR CHOICE

LEFT MARGIN (0-80) ?00
POKE -27607,N - N= 0 TO 80 - YOUR CHOICE

PAGE LENGTH (10-99) ?62
POKE -27606,N - N=10 TO 99 - YOUR CHOICE

INTER-PAGE LENGTH (0-9) ?4
POKE 27605,N - N= 0 TO 9 - YOUR CHOICE

AUTO-NEXT-PAGE (Y/N) ?Y
POKE -27604,89 FOR YES - POKE -27604,78 FOR NO

PAGE NUMBERING (Y/N) ?N
POKE -27603,89 FOR YES - POKE -27603,78 FOR NO

FIRST PAGE NUMBER (1-99) ?01
POKE -27602,N - N= 1 TO 99 - YOUR CHOICE

SUPPRESS FINAL F-F (Y/N) ?N
POKE -27601,89 FOR YES - POKE -27601,78 FOR NO

DOUBLE SPACING (Y/N) ?N
POKE -27600,89 FOR YES - POKE -27600,78 FOR NO

NUMBER OF COPIES (1-9) ?1
POKE -27599,N - N= 1 TO 9 - YOUR CHOICE

WELL THAT COVERS ALL THE DEFAULTS. ONLY CHANGE THE DEFAULTS YOU REQUIRE. TYPE IN RUN AND PRESS RETURN AND YOU SHOULD GET A SCREEN DISPLAY WITH YOUR NEW DEFAULTS. DOUBLE CHECK TO MAKE SURE THEY ARE CORRECT AND IF NOT REPEAT PROCESS ON THOSE THAT ARE NOT.

BE VERY CAREFULL WHEN POKING NUMBERS AS A POKE TO THE WRONG MEMORY LOCATION CAN CORRUPT W.P. IF EVERYTHING IS OK THEN IT'S TIME TO SAVE MODIFIED W.P. TO DISK USING THE FOLLOWING FORMAT :-

BSAVE"WORDPRO",7B80,957F

INSTEAD OF USING WORDPRO YOU CAN USE ANY FILENAME OF YOUR CHOICE. IF YOU HAVE OUR W.P. PATCH3.1 THEN THEN THE FILENAME WORDPROC MUST BE USED. ALL THAT REMAINS NOW IS TO TEST IT OUT.

RESET THE VZ AND BRUN YOUR MODIFIED W.P. AND SELECT PRINT MENU AND THE DEFAULTS SHOULD BE AS PER YOUR SPECS. EVEN THOUGH THE MODS ARE MINOR THE W.P. PRINT MENU IS MUCH EASIER TO USE.

THE WP-PEEK PROGRAM CAN BE USED TO PEEK AT OTHER MEMORY LOCATIONS AS WELL. JUST CHANGE START/END ADDRESSES IN LINE 10. OTHER INTERESTING MEMORY LOCATIONS IN W.P. ARE :-

-28838 TO -28839 *** -28838 TO -28262 *** -28255 TO -28235
-28225 TO -28176 *** -28176 TO -27630 *** -27613 TO -27599
-27472 TO -27395 *** -27330 TO -27268

NEW SERIES ...

vz dos

This is the first in a series where we'll be looking at the VZ disk drive system in some detail. We start by considering what makes up the system, and in the coming months we look at the DOS routines themselves, finishing up with some new routines you can implement yourself. Disk drives are becoming more and more common among VZ users, but even if you don't have one, you should find this series at least interesting.

When you purchased your disk drive system, you were supplied with a disk drive unit, a power supply, a disk controller and a DOS manual.

If you're like me, you were a little dismayed at the DOS manual. Mine was supplied by DSE. It seems that this manual is another 'claytons' attempt at putting together something meaningful. Agreed, it was only designed to make users aware of the DOS commands and how to use them (forget any technical information - this had to come later in the new Technical Manual). What's more, the manual took on a rather non-constructive attitude at times. When discussing the sequential data files, it even goes to the point of saying that the VZ system (of handling data files) is useless:

"...even trying to write anything as complicated as a very short letter becomes difficult and time consuming...many people believe that sequential files are totally useless..."

Perhaps a little more positive approach would have been better! Now that we've got that one out of the way, let's look at the other components in the system.

The disk drive and its controller are completely dependent on one another. The disk drive is designed to read and write data to and from a 5.25" floppy disk. It carries enough electronics to transfer data to and from the disk, control the motor speed, control the movement of the read/write head, and check the write protect status of the disk. It derives its power from the power supply pack. It's actually possible to run two drives from the one power supply, as many of you will have discovered.

The disk controller provides the interface between the computer and the disk drive. It allows the CPU (central processing unit Z80A) to control the functions of the drive mentioned above. It also contains an EPROM which holds the machine code routines which interpret and execute the DOS commands.

Let's take a closer look at what's inside a floppy disk. The 5.25" floppies used by the VZ are an industry standard. Newer PCs are using the smaller, higher capacity 3.5" diskettes. The disk itself is enclosed in a protective cover. The disk is a circular piece of plastic coated with a special formulation of iron oxide, similar to that used on audio cassettes. The centre of the disk is re-enforced (at least on quality disks it is!) with thicker plastic, since this is where the disk drive 'grabs' the disk.

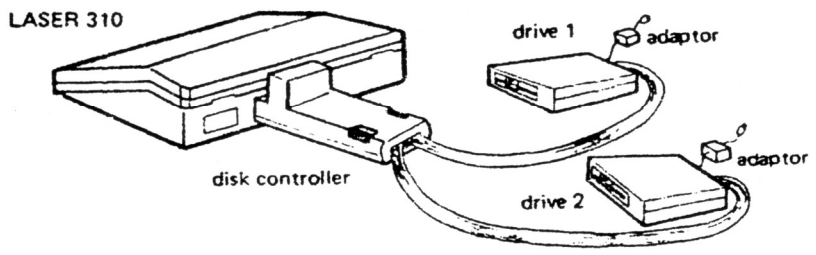
When you buy a pack of disks, the disk is blank. It may have been written to once as a testing (certification) proceedure by the manufacturer, but as far as your VZ goes, there's nothing there. To be able to place data on the disk, the VZ must format the disk in a special way. We'll look at how the disk is formatted in detail later, but for now suffice to say that it is divided into 40 tracks, each track containing 16 sectors.

A track is a circular path around the disk; a narrow strip if you like. Around this strip is written the data to be stored on the disk. You can imagine a piece of audio tape layed in a circle around the disk. As the disk spins, the head (similar to a record/playback head on a cassette recorder) is positioned over a particular track, and the data is read from, or written to, the track. Track 0 is the outermost track, while track 39 is the innermost. Each track is divided into 16 sectors, from 0 to 15. Data is written to the disk in 128 byte 'blocks', each sector holding one 'block'. The sectors are not placed in numeric order around a track, rather an interlace sequence is used, so that the average time to access any particular sector is shorter, and the whole system operates faster. Standard DOS uses the following sequence: 0,11,6,1,12,7,2,13,8,3,14,9,4 ,15,10,5. The new DOS EPROM uses a different, more efficient layout. The numbering and positioning of the tracks and sectors takes place when you first initialise a disk using the INIT command.

You've probably noticed the hole in the jacket of the disk, and the matching hole on the disk itself. This feature is used on many disk drive systems, but not on the VZ! There are two types of diskette; hard-sectored and soft-sectored. The hard-sectored diskette has ten of these holes, while the more common soft-sectored has only one. On other systems, these are used to signal to the drive the start of sector#0. The VZ locates the sector it is looking for by simply spinning the disk and reading each sector number until it finds the one it's looking for.

The small notch in the jacket of the disk is used to write-protect the disk. Inside the drive is a small LED, which directs light through the notch and onto a phototransistor. If the notch is covered, no light passes, and the write-protected signal is set.

Since the VZ does not use an index hole to locate the start of a sector, it needs some way of recognising the start of a sector as it reads data off the disk. This is achieved by starting each sector with a special sequence of bytes. This sequence includes the number of the track in which the sector is located, and the number of the sector itself. Here's the sequence:



GAP 1 (used to identify the start of a new sector)
7 bytes of 80H
1 byte of 00H

IDAM (Identification Address Mark)
4 bytes: FE, E7, 18, C3
Track Number
Sector Number
Checksum

GAP 2 (separates IDAM from data)
80H: 5 bytes
00H: 1 byte
C3H: 1 byte
18H: 1 byte
E7H: 1 byte
FEH: 1 byte

DATA (where the actual data is put)
126 bytes of data
2 bytes indicating next track & sector to read

CHECKSUM (for error checking)
2 bytes

By doing a few simple calculations, you can see that a disk can store a total of 40x16x126= 80640 bytes (this includes directory information).

Having looked at the basics of the disk itself, we'll go on next time with the disk directory and the track map.

FOR SALE

VZ 300 with Datasette and a few tapes.
Whittaker Hamilton is open to offers on this equipment.

I am always sorry to lose members and wish whittaker all the best with his new computer.

Whittaker points out That this VZ 300 will only work on a Monitor and not Television.
A very interesting problem.Anyone with any ideas on this may like to write in.

Anyway,Whittaker's Address is:
58 Brice St
Taupo
Phone (074) 87 117

NEWCOMERS START HERE

Welcome to the confusing world of the microcomputer. First of all, don't be fooled; there's nothing complicated about this business, it's just that we're surrounded by an immense amount of necessary jargon. Imagine if we had to continually say 'numbering system with a radix of 16 in which the letters A to F represent the values ten to 15' when instead we can simply say 'hex'. No doubt soon many of the words and phrases we are about to explain will eventually fall into common English usage.

We'll start by considering a microcomputer's functions and then examine the physical components necessary to implement these functions.

The microcomputer is capable of receiving information, **processing** it, storing the results or sending them somewhere else. All this information is called **data** and it comprises numbers, letters and special symbols which can be read by humans. Although the data is accepted and output by the computer in 'human' form, inside it's a different story — it must be held in the form of an electronic code. This code is called **binary** — a system of numbering which uses only 0s and 1s. Thus in most micros each character, number or symbol is represented by eight binary digits or **bits** as they are called, ranging from 00000000 to 11111111.

To simplify communication between computers, several standard coding systems exist, the most common being **ASCII** (American Standard Code for Information Interchange). As an example of this standard, the number five is represented as 00110101 — complicated for humans, but easy for the computer! This collection of eight bits is called a **byte** and computer freaks who spend a lot of time messing around with bits and bytes use a half-way human representation called **hex**. The hex equivalent of a byte is obtained by giving each half a single character code (0—9, A—F): 0 = 0000, 1 = 0001, 2 = 0010, 3 = 0011, 4 = 0100, 5 = 0101 . . . . E = 1110 and F = 1111. Our example of 5 is therefore 35 in hex. This makes it easier for humans to handle complicated collections of 0s and 1s. The machine detects these 0s and 1s by recognising different voltage levels.

The computer processes data by reshuffling, performing arithmetic on, or by comparing it with other data. It's the latter function that gives a computer its apparent 'intelligence' — the ability to make decisions and to act upon them. It has to be given a set of rules in order to do this and, once again, these rules are stored in **memory** as bytes. The rules are called **programs** and while they can be input in binary or hex (**machine code** programming), the usual method is to have a special program which translates English or near-English into machine code. This speeds programming considerably; the nearer the **programming language** is to English, the faster the programming time. On the other hand, program execution speed tends to be slower.

The most common microcomputer language is **Basic**. Program instructions are typed in at the keyboard, to be coded and stored in the computer's memory. To **run** such a program the computer uses an **interpreter** which picks up each English-type **instruction**, translates it into machine code and then feeds it into the **processor** for execution. It has to do this each time the same instruction has to be executed.

Two strange words you will hear in connection with Basic are **PEEK** and **POKE**. They give the programmer access to the memory of the machine. It's possible to read (PEEK) the contents of a byte in the computer and to modify a byte (POKE).

Moving on to **hardware**, this means the physical components of a computer system as opposed to **software** — the programs needed to make the system work.

At the heart of a microcomputer system is the central processing unit (**CPU**), a single microprocessor chip with supporting devices such as **buffers**, which 'amplify' the CPU's signals for use by other components in the system. The packaged chips are either soldered directly to a printed circuit board (**PCB**) or are mounted in sockets.

In some microcomputers, the entire system is mounted on a single, large, PCB; in others a **bus system** is used, comprising a long PCB holding a number of interconnected sockets. Plugged into these are several smaller PCBs, each with a specific function — for instance, one card would hold the CPU and its support chips. The most widely-used bus system is called the **S100**.

The CPU needs memory in which to keep programs and data. Microcomputers generally have two types of memory, **RAM** (Random Access Memory) and **ROM** (Read Only Memory). The CPU can read information stored in RAM — and also put information into RAM. Two types of RAM exist — **static** and **dynamic**; all you really need know is that dynamic RAM uses less power and is less expensive than static, but it requires additional, complex, circuitry to make it work. Both types of RAM lose their contents when power is switched off, whereas ROM retains its contents permanently. Not surprisingly, manufacturers often store interpreters and the like in ROM. The CPU can only read the ROM's contents and cannot alter them in any way. You can buy special ROMs called **PROMs** (Programmable ROMs) and **EPROMs** (Eraseable PROMs) which can be programmed using a special device; EPROMs can be erased using ultra-violet light.

Because RAM loses its contents when power is switched off, **cassettes** and **floppy disks** are used to save programs and data for later use. Audio-type tape recorders are often used by converting data to a series of audio tones and recording them; later the computer can listen to these same tones and re-convert them into data. Various methods are used for this, so a cassette recorded by one make of computer won't necessarily work on another make. It takes a long time to record and play back information and it's difficult to locate one specific item among a whole mass of information on a cassette; therefore, to overcome these problems, **floppy disks** are used on more sophisticated systems.

A floppy disk is made of thin plastic, coated with a magnetic recording surface rather like that used on tape. The disk, in its protective envelope, is placed in a disk drive which rotates it and moves a **read/write head** across the disk's surface. The disk is divided into concentric rings called **tracks**, each of which is in turn subdivided into **sectors**. Using a program called a **disk operating system**, the computer keeps track of exactly where information is on the disk and it can get to any item of data by moving the head to the appropriate track and then waiting for the right sector to come round. Two methods are used to tell the computer where on a track each sector starts: **soft sectoring** where special signals are recorded on the surface and **hard sectoring** where holes are punched through the disk around the central hole, one per sector.

Half-way between cassettes and disks is the **stringy floppy** — a miniature continuous loop tape cartridge, faster than a cassette but cheaper than a disk system. **Hard disk** systems are also available for micro-computers; they store more information than floppy disks, are more reliable and information can be transferred to and from them much more quickly.

You, the user, must be able to communicate with the computer and the generally accepted minimum for this is the visual display unit (**VDU**), which looks like a TV screen with a typewriter-style **keyboard**; sometimes these are built into the system, sometimes they're separate. If you want a written record (**hard copy**) of the computer's output, you'll need a **printer**.

The computer can send out and receive information in two forms — **parallel** and **serial**. Parallel input/output (**I/O**) requires a series of wires to connect the computer to another device, such as a printer, and it sends out data a byte at a time, with a separate wire carrying each bit. Serial I/O involves sending data one bit at a time along a single piece of wire, with extra bits added to tell the receiving device when a byte is about to start and when it has finished. The speed that data is transmitted is referred to as the **baud rate** and, very roughly, the baud rate divided by ten equals the number of bytes being sent per second.

To ensure that both receiver and transmitter link up without any electrical horrors, standards exist for serial interfaces; the most common is **RS232** (or **V24**) while, for parallel interfaces to printers, the **Centronics** standard is popular.

Finally, a **modem** connects a computer, via a serial interface, to the telephone sytem allowing two computers with modems to exchange information. A modem must be wired into the telephone system and you need Telecom's permission; instead you could use an **acoustic coupler**, which has two obscene-looking rubber cups into which the handset fits, and which has no electrical connection with the phone system — Telecom isn't so uppity about the use of these.